The Evolution of the DARWIN System

Joan D. Walton
NASA
Ames Research Center, MS 269-3
Moffett Field, CA 94035-1000
+1 650 604 2005
jdwalton@arc.nasa.gov

Robert E. Filman
Caelum Research Corporation
Ames Research Center, MS 269-2
Moffett Field, CA 94035-1000
+1 650 604 1250
rfilman@arc.nasa.gov

David J. Korsmeyer
NASA
Ames Research Center, MS 269-3
Moffett Field, CA 94035-1000
+1 650 604 3114
dkorsmeyer@arc.nasa.gov

ABSTRACT

DARWIN is a web-based system for presenting the results of wind-tunnel testing and computational model analyses to aerospace designers. DARWIN captures the data, maintains the information, and manages derived knowledge (e.g. visualizations) of large quantities of aerospace data. In addition, it provides tools and an environment for distributed collaborative engineering. We are currently constructing the third version of the DARWIN software system. DARWIN's development history has, in some sense, tracked the development of web applications. The 1995 DARWIN reflected the latest web technologies—CGI scripts, Java applets and a three-layer architecture—available at that time. The 1997 version of DARWIN expanded on this base, making extensive use of a plethora of web technologies, including Java/JavaScript and Dynamic HTML. While more powerful, this multiplicity has proven to be a maintenance and development headache. The 2000 version of DARWIN will provide a more stable and uniform foundation environment, composed primarily of Java mechanisms. In this paper, we discuss this evolution, comparing the strengths and weaknesses of the various architectural approaches and describing the lessons learned about building complex web applications.

Keywords

WWW applications, DARWIN, wind-tunnel, distributed analysis, collaborative engineering

1. INTRODUCTION

The Internet has transformed distributed applications from unique singularities to ubiquitous commodities. Indeed, many system-building components and subsystems exist (e.g., CGI, Perl, Java, database systems, JavaScript, HTML, DHTML, XML, servlets, CORBA, EJB, and a nearly unbounded number of commercial tools). Building and maintaining applications with these components can be straightforward enough, if the application is simple enough. However, complex distributed applications stress these web development components in ways unanticipated by their creators.

This paper describes the evolution of the DARWIN software system and presents the lessons learned in that evolution with respect to the software engineering of complex web-based systems. Aerospace designers and engineers use DARWIN to understand the results of wind-tunnel testing and numerical model analyses of aircraft designs. Wind tunnel tests place a physical model of a proposed aircraft in an enclosed space, flow 200-600 mile-per-hour winds over the surface of that model, and measure performance attributes such as lift, twist and drag. Over the course of several months, a wind tunnel experiment may measure close to 50,000 instants, each recording up to a thousand variables. Numerical model analyses employ techniques such as computational fluid dynamics to evaluate these physical performance properties from virtual designs. Numerical solutions are computationally resource intensive and can generate gigabyte size results.

DARWIN not only provides distributed, real-time remote access to large volumes of data but also tools for data analysis, visualization, and collaboration. DARWIN also deals with such "real-world" issues as security requirements and the semantic inconsistency endemic to extending legacy systems (i.e. naming conventions and variations in meaning).

We are currently developing the third version of DARWIN. DARWIN's development history has tracked the development of applications on the web. The original version of DARWIN, built in 1995, integrated then "cutting-edge" CGI scripts and Java applets with a Sybase database and Unix file system. Over time, DARWIN evolved to incorporate the latest advances in browser technology. The current operational system, version 2 of DARWIN, also extensively uses Java/JavaScript and Dynamic HTML to add features and responsiveness to the user interface. The multiplicity of mechanisms in DARWIN 2 makes it difficult to maintain—the components themselves evolve as DARWIN 2's users desire increased functionality and as the component manufacturers release new versions. The current web-architecture nicely handles *access* to the aeronautics data for distributed users, but in the future, DARWIN must also handle *retrieval* of the data from distributed databases.

We are now in the process of designing version 3 of DARWIN. We are guided by the maxim "less is more." That is, to produce a more effective and maintainable system, we are driving to reduce the variety of development mechanisms. DARWIN 3 will be entirely Java-based: Java clients (applet or application) talking to Java servlets communicating with data sources (databases and applications) via Java/CORBA services. We believe this will greatly simplify the overall development process.

2. BACKGROUND

The DARWIN system allows its users to access aerospace data through a collection of displays and also to perform various analysis functions. In a typical session, a DARWIN user might perform the following tasks:

- **Establish security context**

  After connecting to the DARWIN web server with a Netscape web browser, the user logs in with her name and password. Our facility serves a national community of aerospace designers. Such customers are unenthusiastic about granting competitors access to data on their proprietary designs. To address these concerns, all communications with the web server take place over secure http, and the user is authenticated by IP address in addition to password.

- **Browse**

  The DARWIN home page provides overview screens for the wind tunnel tests and computational fluid dynamics solutions available (based upon user ID) within the system. From these screens the user can see what tests are in the system, get basic information about the tests, and check the test's bulletin boards where messages and files are exchanged. Figure 1 shows an example DARWIN home page with wind tunnel tests displayed.

Figure 1: DARWIN Home page with wind tunnel tests displayed.

An in-depth look at the data can be performed by creating a dataset *review*. The user selects the data of interest via the browsing screens and launches the review screen. The review consists of two types of tabular displays (data summary table and sequence table) and a set of plots. The user can choose which variables are displayed in the tables and in the plots. Additional data points can be added to a review by invoking the query screen and searching for points of interest. Figure 2 shows a three-dimensional plot of some wind-tunnel data.

Having selected the data to review and configured the tables and plots, the user can save her work into a DARWIN dataset. This structure retains enough information for DARWIN to recreate the review screen at a later time. Datasets are managed in a database on a per user basis.

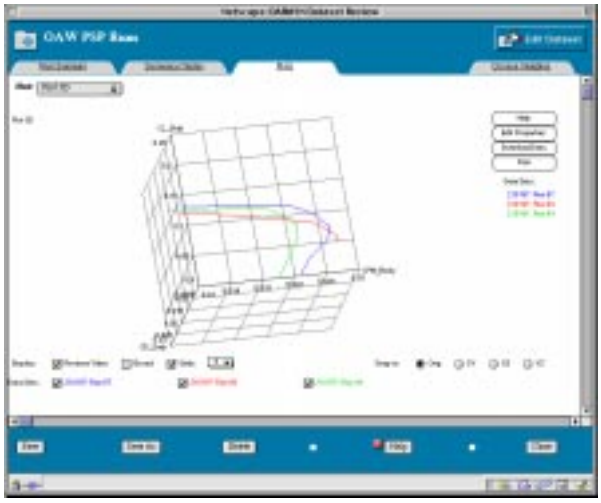

Figure 2: The DARWIN Review Screen showing a three-dimensional plot

- **Interact with the world**

  DARWIN is not just a database retrieval and presentation mechanism. It also provides two kinds of interaction: real-time monitoring of in-progress testing and collaboration with colleagues. While a wind tunnel test is in progress, users can monitor its progress via the *live screen*. The live screen has the same tables and plots as the review plus current status indicators, message board, and shared files "shelf." The tables and plots are updated every 20 seconds to show the latest collected data. Figure 3 shows the live screen in operation.

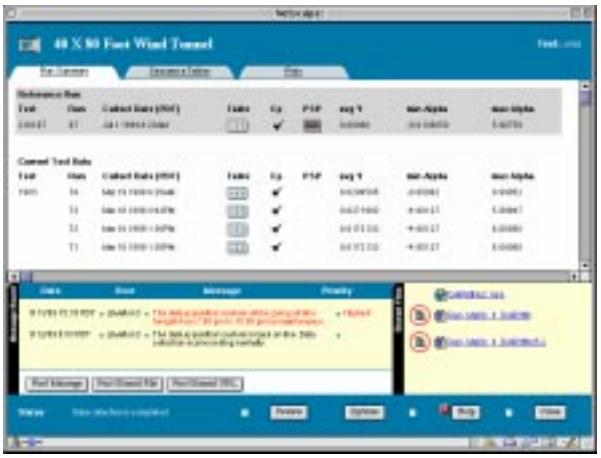

Figure 3: Live Screen

DARWIN also provides several collaboration mechanisms for keeping in touch with team members. Users can post messages and files associated with a particular test and can define mail groups for sending group email and tracking the threaded conversations.

Wind tunnel test data are grouped into "time instants" or *points.* For each point, the wind tunnel data acquisition system collects a large volume of information about conditions in the tunnel and on the model for a given configuration. For example, pressure taps on the model can reveal detail about the air pressure at specific spots on the model, and pressure sensitive paint can produce a continuous pressure map across the model's surface. These measurement systems produce files containing the results. Pressures measured at specific locations are stored in text files. Pressure sensitive paint results are recorded with a camera, so those results are stored in image files. DARWIN deals with a variety of file types, both text and binary.

The large volume of data associated with actual tests has led us to a design incorporating a "meta" database. This database is a relational database that stores information *about* the data points. The meta-database holds both data applicable to the experiment as a whole and variables on which users are likely to want to search. For example, tunnel conditions such as wind speed, temperature and angle of attack of the model are data applicable to the entire experiment. Likewise, overall lift and drag apply to the model as a whole. Because users may want to find, for example, the data point with the greatest drag, that information is also included in the meta-database. The data contained in files produced by specialized measurement systems are considered detail data. The meta-database stores the locations of these files, and the time points with which they are associated, so that the files can be retrieved as needed.

Although DARWIN is an aerospace application, it is also a generic application. What DARWIN does is (1) present to distributed users large volumes of both numeric and image data gathered from multiple sources and (2) provide visualization tools for examining the data, collaboration tools for working cooperatively with the data, and real-time mechanisms for interacting with ongoing activities. The DARWIN architecture and experience thus generalizes across domains with similar (and simpler) problems.

3. DARWIN 1

The first version of DARWIN was developed in 1995 in the infancy of web applications. The architecture combined a classic three-tier approach with a fourth "file system" layer, as is illustrated in Figure 4.

Figure 4: The three+ tier architecture of DARWIN 1.

The client user interface was based on the beta release of Netscape Navigator 1.0 and was limited by that platform's lack of mechanisms for dynamic effects. The screens were pure HTML. As new features such as frames, JavaScript, and Java applets were added to Navigator, we rapidly incorporated them into DARWIN to improve the user experience. Whenever possible, we moved interfaces updates to the client, thereby reducing communication and server load, and providing the user with a more dynamic interface. Thus, instead of producing static images on the server to display plots of the data, a two-dimensional plotting applet was developed in-house. The applet had the advantage of being an encapsulated program running on the client, so, for example, it responded to user imperatives to zoom or remove display points locally. JavaScript functions were used toward the same end: giving the interface the power to modify itself without having to request new HTML pages from the web server.

When unavoidable, web pages were constructed by CGI scripts on the web server. Like many contemporary applications (and currently), the scripts were written in Perl. A Perl module was employed for communicating with the Sybase database, and the SQL statements for retrieving information from the meta-database were embedded directly in the scripts. When the user requested information that resided in a data file, the scripts would retrieve the location of the file from the meta-database, fetch the file from the indicated machine (via http), and build a display to present the file's contents. The files dealt with at that time were often formats used only at our wind tunnel facility.

DARWIN 1 was the first system to allow remote access to wind tunnel data. Even with the initial user interface enhancements afforded by the availability of Java applets and JavaScript, the client was still very light and fast. The database contained data from only a few experiments, and its response time was quick. DARWIN 1 provided few browsing tools, so the amount of system code was small and easy to track and maintain. The system was deployed and put to use by engineers doing wind tunnel experiments. The users liked and approved of the system, and the project received an award from NASA for achieving broad industry acceptance of its information system concepts. Once the engineers got a

feel for what could be done with DARWIN, they began to request more features, notably the ability to compare data across experiments and monitor tests in progress. They also wanted avenues for collaborating with their peers. Rather than shoehorn the new features into the original interface design, we did a complete redesign and developed DARWIN version 2.

4. DARWIN 2

By late 1997, a whole new generation of tools for enhancing browser applications had become available. JavaScript had matured into a full-featured language, and dynamic HTML and stylesheets allowed increased control of the appearance and behavior of web pages. These new tools were enthusiastically applied in the development of DARWIN 2. The result was a web application that looked and behaved more like a stand-alone program than like traditional web pages. Stylesheets allowed consistent control of colors and fonts and provided exact positioning of graphical elements. DHTML provided more tools for making the interface dynamic and furthered the goal of minimizing calls to the web server.

DARWIN 2 presented a comprehensive set of browsing functions and allowed users to interact with their colleagues at remote sites and monitor tests in progress. The new monitoring *live screen* presented the most recent data in tables and plots that were updated as new data came in. (Pushing data to a browser is not a web primitive. The updating was accomplished with an inconspicuous frame at the bottom edge of the screen that loaded an updater CGI script every 20 seconds. When the script detected new data, it would reload the appropriate displays.) Live screen also displayed a simple message board where users could post notes to each other, and a shared "shelf" where files could be posted and downloaded.

The live screen quickly became popular with our users. Engineers at remote locations would leave the screen running all day to stay abreast of the test's progress. If they had questions or comments about what they saw, they could phone the test engineer at the tunnel. The shared file shelf proved useful for securely transferring files from the tunnel. thus, less secure connections to the tunnel systems, such as ftp, could be disabled while the engineers retained access to the data they required.

In addition, DARWIN 2 introduced the concept of "studies." Users could modify the standard views into the database and save them along with associated files and hyperlinks into a user-specific virtual file system. This allowed client location independence for DARWIN access. It did not matter where, in an approved IP range, the user accessed DARWIN because all of the personalized studies and associated files were always available.

Despite the advances in web components, limitations of the browser and the web architecture required some design compromises:

- **Download delays**

  The user interface for the review screen in DARWIN 2 emulated a set of index tabs cards for various information displays. The initial implementation loaded all the displays on every client call. As some of the displays were fairly complex, this proved too time consuming. We were able to create more responsive screens with DHTML by putting the displays into "layers" and then hiding or showing the layers appropriately. However, all the layers still had to be built before the screen could be used, producing a front-loading delay. The user had to wait, sometimes for a significant amount of time, for the screen to finish loading; but once done, the user could switch between displays quickly and easily. In addition, the database was now populated with several years' worth of experimental data, so queries were slower as well.

- **Architectural complexity**

  DARWIN was now a complex construction of components of various types—mainly Java applets, JavaScript objects, and CGI scripts—and these components all needed to communicate with each other. Anytime a query was directed at the database, a CGI script had to be called to make the database connection, execute the query and return the results. Communicating those results to the other components without forcing the whole screen to reload was tricky business as was retrieving client-side state and saving it to the server.

  Using the best web component for each task (e.g., DHTML for graphical precision, JavaScript for control of DHTML elements, Java for plotting, and CGI scripts for interacting with the database) simplified creating the system but produced a difficult-to-maintain monster. Tracking the successive releases of the browser and components turned into a major task. In the future, once the original developers had moved on, we anticipate it would be difficult to find maintenance staff with a large variety of skills to handle all these different packages.

- **Client load**

  Integration of multiple browser components had made the client system heavy. In particular, rendering of DHTML and the large JavaScript components stressed the browser and increased speed and memory requirements on the client machine.

Table 1 illustrates the relative complexity of DARWIN 1 and DARWIN 2. Version 2 does a lot more, but there is five and a half times as much code in the system, with too many different kinds of components. The Perl/CGI number for DARWIN 2 includes 14,679 lines of imported packages.

	DARWIN 1	DARWIN 2
Perl/CGI	8,693	52,406
HTML	2,161	3,436
JavaScript	130	16,398
Java	7,306	20,201
C, C++		9,201
Gif's, by instance	25	91
Total lines	18,315	101,642

Table 1: The comparative size of DARWIN 1 and 2

DARWIN 1 took 18 months for one programmer to develop. DARWIN 2 took three programmers and a database designer another 18 months to complete. The approach used for development of these applications was (1) build a solid, architectural design, (2) create a prototype of the system, and (3) engage in an extended iterative design-and-code phase where feedback from the users was incorporated promptly into the system.

5. DARWIN 3

Success begets demands for greater functionality. The requirements for DARWIN 3 are significantly more demanding than the requirements for the previous version of DARWIN in several areas:

- **Remotely located data sources**

  DARWIN 1 and 2 provided access for geographically distributed users to a centrally located meta-database and data repository. With DARWIN 3, there will be multiple distributed meta-databases and data repositories. Dealing with distributed databases increases system complexity and network latency.

- **Distribution of user management tasks**

  In DARWIN 1 and 2, administrative tasks such as creating new user accounts and groups and setting access privileges were performed centrally. As tunnels from other facilities join the DARWIN system, the people in charge of those tunnels should naturally be in control of who gets access to their data and also be allowed to add their own people to the system. Administration must become distributed, and the not-necessarily-identical security policies of these different domains will need to be supported.

- **Full-featured collaboration tools**

  DARWIN 1 provided one view into the data. Users could customize that view, but could not save it or create multiple views. With DARWIN 2, users could save their views, group them into "studies" and associate files, such as spreadsheets or images, with those studies. In DARWIN 3, the users need to be able to share their work with colleagues by allowing access to the views they have created while not violating the access control rules established by the DARWIN administrators.

The additional complexity of these tasks coupled with the already over-taxed browser-based architecture of DARWIN 2 make re-engineering DARWIN a necessity. To reduce complexity of the code, we are developing the architecture in Figure 5.

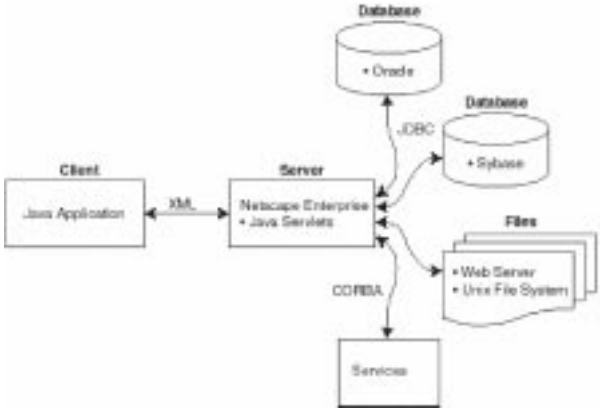

Figure 5: The architecture of DARWIN 3

Where DARWIN 2 incorporated HTML, JavaScript, DHTML, and Java applets all running in a browser, the DARWN 3 client is a single, unified, stand-alone Java application. The decision to give up the browser is a difficult one. On one hand, browsers are ubiquitous and freely available, which makes client installation fairly straightforward. Browsers also handle certain security issues automatically. On the other hand, every time Netscape released a new version of Navigator, the system would break or perform in unexpected ways. We considered making the DARWIN client one large applet, but were hindered by the lag time between the introduction of new versions of Java and the incorporation of those versions into the Netscape browser. At the time of this writing, implementing DARWIN as an applet would either preclude our using the Java Foundation Classes or require our users to install the Java 2 plug in, both undesirable outcomes. Implementing DARWIN 3 as a Java application will require more effort to perform client installations, but we are hopeful that one of the commercial tools available for this purpose will ease that process.

On the server side, Perl CGI scripts will be replaced with Java servlets. Servlets will let us naturally manage sessions and track the session state. In addition, Java classes are available to connect to databases with JDBC, make CORBA requests, or access LDAP directories. CGI scripts spawn a process for every http access and each database connection made within a CGI script must login and log out of the database. Servlets do not spawn child processes and can access an open connection to the database, thus saving significant processing time.

Keeping consistent policies among a collection of distributed services can itself be a problem. We plan on using the Object Infrastructure Framework (OIF) [2] to provide a dynamic mechanism for inserting policies on security, reliability, quality of service, accounting, debugging, performance measurement and configuration management into the service network. OIF generates Java injected behaviors on the proxies of distributed services. This allows us to localize, for example, the security requirements of each distributed dataset to the particular requirements of that dataset. With this system we can fine tune security to the content of the results or simply impose, down the road, an accounting policy on resource usage.

DARWIN 3 will be developed using the same process as was used with DARWIN 1 and 2, preceded by a "tool experimenting" phase. Conventional application prototyping focuses on testing the user interface and the performance of the system. In the rapidly changing domain of web applications, however, significant time must also be spent on experimenting with new tools and components, learning how to use them, and determining whether they are effective solutions to the problem at hand.

The DARWIN 3 system will provide its users with the tools to perform many more data analysis and collaboration tasks than were possible with DARWIN 1 or 2. Nevertheless, because all the tools will be written in the same language, managing the software development process will be simpler. Similarly, we can dispense with the myriad instances of special code for linking up components: Java classes integrate seamlessly. We anticipate that this simplification will allow us to focus our effort on modeling the complex domain and inventing the appropriate interfaces for exhibiting that domain, not fighting the inconsistencies of the programming environment.

6. RELATED WORK

Within NASA several systems have been developed with remote access to aerospace data. In 1994, NASA Ames Research Center developed a system called remote access wind tunnel (RAWT) [4]. In 1995, NASA Glenn Research Center modified this concept to create remote access control room (RACR). Both systems were Unix only and used a commercial whiteboard program called InPerson for Silicon Graphics machines and X-windows. No database of information was developed, as the emphasis was remote access and collaboration.

Similarly, several data and documentation tools were developed at NASA Langley Research Center and NASA Ames. These were PrISM, and ADAPT at Langley and PostDoc at Ames. PrISM collected the wind tunnel data into a database on a test by test basis and provided a robust query capability to download the results to the user. No presentation or naming consistency was developed under PrISM. ADAPT and PostDoc were similar in that they were early web-based document management systems. ADAPT focussed upon creating secure access to encrypted documents. Any data or documents were stored as an encrypted file with the user required to have a decryption helper application for the web browser to decrypt data to the desktop. PostDoc emphasized capturing and translating documents into Adobe PDF files to broad access to many platforms. PostDoc only addressed security through a user id and password scheme.

As DARWIN became operational, the U.S. Air Force's Arnold Engineering and Development Center held a meeting in the spring of 1997 to assess NASA's aerospace data management tools. DARWIN was selected as the desired model for the USAF to emulate and a project called Integrated Test and Information System (ITIS) was initiated. It is building a DARWIN compatible meta-database and an in-house set of analysis tools.

Of course, use of the Internet for database access, collaboration, and real-time monitoring has many antecedents. For example, Evans and Rogers [1] report on using Java applets and CORBA to reimplement (parts of) an existing multi-user WWW application, replacing the existing CGI scripts. They found the applet/CORBA combination to be better at performing client-side applications, to be easier to maintain, to be simpler to program (because of the ability to retain server-side state), to be more straight forward to deploy, and to provide greater responsiveness.

Ly [5] describes Netmosphere ActionPlan, a web-enabled project management product. Architecturally, ActionPlan is a pair of client applets linked to a Java-language server. A key element of Netmosphere was the real-time, selective notification mechanism for keeping information synchronized.

Itschner, Pommerell and Rutishauser [3] report on the GLASS system, which uses internet technology to monitor remote embedded systems. GLASS proxies accumulate data from embedded system monitoring devices and store this information on the database of a server. Client applications, running in browsers with Java applets, retrieve this data through CGI scripts on the server.

Tesoriero and Zelkowitz [6] have developed the WebME system, which uses a mediating query processor, metadata database, and wrappers on the information repositories to direct queries to the appropriate databases.

7. DISCUSSION

The common technology of the Internet and world-wide-web markup languages, internet protocols, servers, Java, CORBA, and so forth have taken the task of building complex distributed applications from expeditions to outings. With DARWIN, in a few short years we have transformed access to data from wind tunnel experiments from a slow and cumbersome process to an immediate, real-time, interactive, collaborative experience. This has been accomplished by relying on a large variety of Internet technologies. Like kids in the candy store, we have applied each tool for its particular strength. Moving forward in this process, we see that a long-term maintainable system requires fewer mechanisms. Although individually, technologies such as DHTML, browsers, and CGI scripts simplify specific tasks, integration and evolution requirements argue less is more, and that being closer to the programming language level (Java), particularly with a network-aware language like Java, will make for a sustainable environment.

8. ACKNOWLEDGMENTS

The authors would like to thank the DARWIN development team for building quality applications and making the project a success.

9. BIOGRAPHIES

The authors are Research Scientists in the Computational Sciences Division of the NASA Ames Research Center. Joan D. Walton was the original programmer on the DARWIN project and now leads a team of eight developers working on distributed, internet-based applications. Before coming to NASA five years ago, she built analytical software for

biotech and medical instruments. Ms. Walton received her M. S. in Medical Information Sciences from Stanford University.

Robert E. Filman is working on frameworks for developing distributed applications. He has published in the areas of software engineering, distributed computing, network security, programming languages, artificial intelligence, and human-machine interface. Dr. Filman received his Ph. D. in Computer Science from Stanford University.

David J. Korsmeyer was one of the originators of DARWIN in 1995. He is now a program manager for NASA's Information Technology Base Program and the Chief Information Technology Architect for NASA's Science and Engineering Infrastructure. Dr. Korsmeyer has been at NASA for nine years and received his Ph. D. in Aerospace Engineering from the University of Texas at Austin.

10. REFERENCES

[1] Evans, E. and Rogers, D. Using Java Applets and CORBA for Multi-User Distributed Applications. *IEEE Internet Computing 1*, 3 (May 1997) 43-55.

[2] Filman, R. E., Barrett, S., Lee, D. D., and Linden, T. Inserting Ilities by Controlling Communications. To appear in *Comm. ACM.*

[3] Itschner, R., Pommerell, C., and Rutishauser, M. GLASS: Remote Monitoring of Embedded Systems in Power Engineering. *IEEE Internet Computing 2*, 3 (May 1998) 46-52.

[4] Koga, D. J., Schreiner, J. A., Buning, P. G., Gilbaugh, B. L., and George, M. W. Integration of Numerical and Experimental Wind Tunnel (IofNEWT) and Remote Access Wind Tunnel (RAWT) Programs of NASA. 19th AIAA Advanced Measurement and Ground Testing Technology Conference, New Orleans, LA, June 1996, AIAA-96-2248.

[5] Ly, E. Distributed Java Applets for Project Management on the Web. *IEEE Internet Computing 1*, 3 (May 1997) 21-27.

[6] Tesoriero, R., and Zelkowitz, M. A Web-based Tool for Data Analysis and Presentation. *IEEE Internet Computing 2*, 5 (September 1998) 63-69.